Problem 1:

Markov processes with an infinite number of states arise in queuing theory. For instance, imagine we have a queue (of jobs, of messages arriving at a node of the internet, whatever...). At each time step, a new item arrives in the queue with probability p, and an item is removed from the queue with probability q. No more than one item arrives or is removed from the queue in each time step. We model the queue as a Markov process in which the state s_i is the state in which there are i elements in the queue. Thus, there are an infinite number of states, $s_0, s_1, s_2, \ldots$.

a. For a state $s_i, i \geq 1$, draw the directed edges going into and out of s_i, and the states to which they go, labelled with the appropriate transition probability.

Solution: Rather than a nice drawing, we show a state matrix for s_i. Since this is a matrix for s_i, we show only the transitions that involve s_i, either in or out:

	s_{i-1}	s_i	s_{i+1}
s_{i-1}		$p(q-1)$	
s_i	$q(p-1)$	$pq + (1-q)(1-p)$	$p(1-q)$
s_{i+1}		$q(p-1)$	

b. Assume $p < q$. Let

$$r = \frac{p(1-q)}{q(1-p)} \tag{1}$$

Verify arithmetically that the probability distribution in which the probability of s_i is $(1-r)r^i$ is a stationary distribution for this Markov process.
Solution: We must show that $\mathbf{u}^{(1)} = \mathbf{uP} = \mathbf{u}$, where $\mathbf{u}$ is the initial probability distribution vector, and $\mathbf{P}$ is the transition matrix, defined as follows:

$$\mathbf{u} = \{\ldots, (1-r)r^{i-1}, (1-r)r^i, (1-r)r^{i+1}, \ldots\} \tag{2}$$

$$\mathbf{P} = \begin{pmatrix} \ddots & 0 & \cdots \\ \vdots & \vdots & \cdots \\ \cdots & p(1-q) & \cdots \\ \cdots & pq + (1-p)(1-q) & \cdots \\ \cdots & q(1-p) & \cdots \\ \cdots & 0 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{3}$$

Therefore, by the row times column rules of vector-matrix multiplication, s_i of $\mathbf{uP}$ is as follows:

$$s_i = (1-r)r^{i-1} \cdot p(1-q) + (1-r)r^i \cdot (pq + (1-p)(1-q)) + (1-r)r^{i+1} \cdot q(1-p) \tag{4}$$

$$= (1-r)r^i \left(\frac{1}{r} \cdot p(1-q) + pq + (1-p)(1-q) + r \cdot q(1-p) \right) \tag{5}$$

$$= (1-r)r^i \left(\frac{q(1-p)}{p(1-q)} \cdot p(1-q) + pq + (1-p)(1-q) + \frac{p(1-q)}{q(1-p)} \cdot q(1-p) \right) \tag{6}$$

$$= (1-r)r^i \big(q(1-p) + pq + (1-p)(1-q) + p(1-q) \big) \tag{7}$$

$$= (1-r)r^i (q - pq + pq + 1 - p - q + pq + p - pq) \tag{8}$$

$$= (1-r)r^i \tag{9}$$

Thus, we can see that $\mathbf{u}^{(1)} = \mathbf{u}$, which means that this is a stationary distribution for this Markov process.

c. If $p > q$, does there exist any stationary probability distribution?

Solution: Let $\mathbf{u} = \{P_0, P_1, \ldots\}$ represent a stationary probability distribution. Thus, it must satisfy the following equation:

$$P_i = P_{i-1} \cdot p(1-q) + P_i \cdot (pq + (1-p)(1-q)) + P_{i+1} \cdot q(1-p) \tag{10}$$
$$P_{i+1} \cdot q(1-p) = P_i - P_{i-1} \cdot p(1-q) - P_i \cdot (pq + (1-p)(1-q)) \tag{11}$$
$$P_{i+1} \cdot q(1-p) = P_i \cdot (1 - (1-p-q+2pq)) - P_{i-1} \cdot p(1-q) \tag{12}$$
$$P_{i+1} \cdot q(1-p) = P_i \cdot (p - pq + q - pq)) - P_{i-1} \cdot p(1-q) \tag{13}$$
$$P_{i+1} = P_i \cdot \left(\frac{p(1-q)}{q(1-p)} + \frac{q(1-p)}{q(1-p)}\right) - P_{i-1} \cdot \frac{p(1-q)}{q(1-p)} \tag{14}$$
$$P_{i+1} = P_i(r+1) - P_{i-1}r \tag{15}$$

Now, we **guess** that this recurrence relation for P_{i+1} can be satisfied if P_i is of the form $P_i = c_1\alpha^i + c_2$, and we will verify that that is in fact the case, and then solve for α:

$$c_1\alpha^{i+1} + c_2 = (c_1\alpha^i + c_2)(r+1) - (c_1\alpha^{i-1} + c_2)r \tag{16}$$
$$c_1\alpha^{i+1} + c_2 = c_1\alpha^i r + c_2 r + c_1\alpha^i + c_2 - c_1\alpha^{i-1}r - c_2 r \tag{17}$$
$$c_1\alpha^{i+1} = c_1\alpha^i r + c_1\alpha^i - c_1\alpha^{i-1}r \tag{18}$$
$$\alpha^{i+1} = \alpha^i r + \alpha^i - \alpha^{i-1}r \tag{19}$$
$$\alpha^2 = \alpha(r+1) - r \tag{20}$$
$$\alpha^2 - \alpha(r+1) + r = 0 \tag{21}$$

We can simply use the quadratic equation to solve the last equation:

$$\alpha^2 - \alpha(r+1) + r = 0 \tag{22}$$
$$\alpha = \frac{(r+1) \pm \sqrt{(r+1)^2 - 4r}}{2} \tag{23}$$
$$\alpha = \frac{(r+1) \pm \sqrt{r^2 + 2r + 1 - 4r}}{2} \tag{24}$$
$$\alpha = \frac{(r+1) \pm \sqrt{r^2 - 2r + 1}}{2} \tag{25}$$
$$\alpha = \frac{(r+1) \pm \sqrt{(r-1)^2}}{2} \tag{26}$$
$$\alpha = \frac{(r+1) \pm (r-1)}{2} \tag{27}$$

Thus, taking both the positive and negative solutions, we see:

$$\alpha = \frac{(r+1) \pm (r-1)}{2} \tag{28}$$
$$\alpha = r \tag{29}$$
$$\alpha = 1 \tag{30}$$

But, $r > 1$ because $p > q$. Thus, because any probability distribution must sum to 1, neither of these solutions to the recurrence relation can be a probability distribution, because $\sum_{i=0}^{\infty}(c_1\alpha^i + c_2)$ diverges to infinity for both values of α. Thus, there is no stationary distribution possible if $p > q$.

Problem 2:

- **Definition:** An *independent set* is a set $I \subseteq V$ of vertices such that for all $u, v \in I$, $(u, v) \notin E$.
- **Definition:** The *independent set decision problem* is as follows: Given a graph G, is there an independent set I of vertices of size at most k?

a. Show that INDEPENDENT SET is NP-COMPLETE.
Solution: Our solution consists of two parts. We first show that INDEPENDENT SET (IS) is in NP and then we show that IS is NP-HARD.

- NP: In order to show that IS $\in$ NP, we must show that there exists a polynomial time *witness* algorithm that takes an instance of the problem and a certificate as parameters, and verifies that the certificate is a *yes* instance of the particular inputed problem. Thus, our instance is an unweighted graph G, and our certificate is a set of vertices. Our algorithm performs 2 steps:
  1. Checks that the all vertices in the certificate are vertices in the graph.
  2. Checks that there are no edges between any two vertices in the certificate.

  This algorithm runs in $O(V + E)$, and is thus clearly polynomial time. Thus, IS $\in$ NP.
- NP-HARD: In order to show that IS $\in$ NP-HARD, we will reduce *from* CLIQUE *to* IS. Our reduction consists of demonstrating a polynomial time conversion of an instance of CLIQUE to an instance of IS, and an if-and-only-if proof that a *yes* instance of CLIQUE maps to a *yes* instance of IS and vice versa.
  1. We will convert an instance of CLIQUE to an instance of IS in the following manner. An instance of CLIQUE is a graph G and an integer k. We will construct G^c, the complement of G, and pass G^c, k to IS.
  2. Now we will show that a *yes* instance of CLIQUE maps to a *yes* instance of IS and vice versa.
     - $\implies$ Assume G is a *yes* instance of CLIQUE, or assume that there exists a clique C of size k in G. Thus, for any $u, v \in C$, $(u, v) \in E$. Thus, $(u, v) \notin E^c$. Thus, the vertices in C form an indepedent set in G^c. Thus, G^c is a *yes* instance of IS.
     - $\impliedby$ Assume G^c is a *yes* instance of IS, or assume that there exists an independent set I of size k in G^c. Thus, for any $u, v \in I$, $(u, v) \notin E^c$. Thus, $(u, v) \in E$. Thus, the vertices in I form a clique in G. Thus, G is a *yes* instance of CLIQUE.

Thus, since we have shown that IS is in both NP and NP-HARD, we have shown that IS $\in$ NP-COMPLETE.

b. When G is a tree (not necessarily binary), is this problem still NP-COMPLETE?
Solution: No.
We will store two values with every vertex, $\max_{\text{in}}$ and $\max_{\text{out}}$, referring to the size of the largest indepedent set possible if that particular vertex is either *in* or *out* of the independent set. Our polynomial-time algorithm to find a maximum-cardinality independent set is thus constructed as follows.

1. DFS down to a leaf. Let $\max_{\text{in}} = 1$ and $\max_{\text{out}} = 0$.
2. Begin to recurse back out of tree. For each interior node, $\max_{\text{in}}$ and $\max_{\text{out}}$ are defined as follows:

$$\max_{\text{in}} = 1 + \sum_{\text{children}} \max_{\text{out}} \tag{31}$$

$$\max_{\text{out}} = \sum_{\text{children}} \max_{\text{in}} \tag{32}$$

3. Thus, to determine the largest independent set, we simply take the $\text{MAX}(\max_{\text{in}}, \max_{\text{out}})$ of the root.

This algorithm clearly runs in $O(V + E)$ time because we are only augmenting DFS with a few arithmetic operations per step.

Problem 3:

a. Formulate the optimization version of INDEPENDENT SET as a ZERO-ONE INTEGER program.

Solution:

Maximize:

$$\sum_{v \in V} x_v \tag{33}$$

Subject to:

$$x_u + x_v \leq 1 \qquad \forall (u, v) \in E \tag{34}$$

$$x_v \geq 0 \qquad \forall v \in V \tag{35}$$

$$x_v \in \{0, 1\} \qquad \forall v \in V \tag{36}$$

Thus, the x_v variables are either 1 or 0, depending on whether or not the vertex in or out of the independent set, and no two adjacent vertices can both be in the set.

b. Ignoring the constraint that the solution must be an integer, this problem becomes a linear program. Write down the dual of this linear program.

Solution:

Minimize:

$$\sum_{(u,v) \in E} y_{uv} \tag{37}$$

Subject to:

$$\sum_{v \in V : (u,v) \in E} y_{uv} \geq 1 \qquad \forall u \in V : \exists v \in V \text{ with } (u, v) \in E \tag{38}$$

$$y_{uv} \geq 0 \qquad \forall (u, v) \in E \tag{39}$$

$$\tag{40}$$

c. Now constrain the solution to the dual problem to be an integer, producing another special case of INTEGER PROGRAMMING. In terms of the graph G, this integer program is asking for a subset of some elements of G, which is optimal in some sense. What is the subset and how is it optimal?

Solution: This linear program is asking us to minimize a set of edges such that every vertex with outgoing edges is incident in at least one edge in the set. This is an *edge cover*, similar to a vertex cover.

Problem 4:

- BIN-PACKING: The *optimization* version of BIN-PACKING is defined as follows:
  - **Input:** An integer T and a list of integers $X = (x_1, x_2, \ldots, x_n)$ where $x_i \in [0, T]$.
  - **Output:** A partition of X into a minimum number of sublists, such that each sublist sums to at most T.
- BIN-PACKING: The *decision* version of BIN-PACKING is defined as follows:
  - **Input:** An integer T, an integer k, and a list of integers $X = (x_1, x_2, \ldots, x_n)$ where $x_i \in [0, T]$.
  - **Output:** *Yes* if there exists a partition of X into at most k sublists, such that each sublist sums to at most T, and *no* otherwise.

- PARTITION: The *decision* version of PARTITION is defined as follows:
  - **Input:** A list of integers $X = (x_1, x_2, \ldots, x_n)$
  - **Output:** *Yes* if there exists a partition of X into two lists which sum to the same value, and *no* otherwise.

Show that there cannot exist any polynomial-time approximation algorithm for BIN-PACKING with approximation ratio less than 3/2 (unless P = NP).

Solution: Consider the following proof. Assume there exists a polynomial-time approximation algorithm A with an approximation ratio less then 3/2. This implies that if L^* is the optimal number of lists, than A will return a number of lists L such that $L \leq 3/2L^*$. We will show that A can be used to solve PARTITION $\in$ NP-COMPLETE in polynomial time, thus implying that P = NP. Our proof is as follows:

1. We will convert an instance of PARTITION to an instance of the decision version of BIN-PACKING as follows. For $X = (x_1, x_2, \ldots, x_n)$, let $T = \sum_{i=1}^{n} x_i$. Let $(\lfloor T/2 \rfloor, 2, X)$ be the corresponding instance of BIN-PACKING.

2. If there exists a partition in X, then clearly X can be packed into two bins of size $\lfloor T/2 \rfloor = T/2$ because T is even. But if X is a *no* instance of PARTITION, then the smallest number of bins required to pack X would be 3, because at least one element would not fit in one of the bins of size $\lfloor T/2 \rfloor$. Note that $2 \cdot 3/2 = 3$.

3. Thus, given an instance of PARTITION, we convert it to an instance of BIN-PACKING as above, and then pass that instance to A, our assumed less-than 3/2-approximation algorithm. If a partition exists, *A must return that partition* because the next best solution possible is exactly $3/2L^*$. Since A runs in polynomial-time, we will have solved PARTITION in polynomial-time, which implies that P = NP.

Thus, we have shown that if a less-than 3/2-approximation algorithm exists, then P = NP. Notice that we have **not** shown that it is impossible for such an approximation algorithm to exist– just that a very surprising complexity result would occur if it does.